50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. DCA/SW/MT-88/001i	2.	3. Recipient's Accession No.

4. Title and Subtitle	5. Report Date
Defense Communications Agency Upper Level Protocol Test System File Transfer Protocol Remote Driver Specification	May 1988
	6.
7. Author(s)	8. Performing Organization Rept. No
9. Performing Organization Name and Address Defense Communications Agency Defense Communications Engineering Center Code R640 1860 Wiehle Ave. Reston, VA 22090-5500	10. Project/Task/Work Unit No. 11. Contract(C) or Grant(G) No. (C) (G)
12. Sponsoring Organization Name and Address	13. Type of Report & Period Covered FINAL 14.

15. Supplementary Notes

For magnetic tape, see: ADA 195128

16. Abstract (Limit: 200 words)

This document is part of a software package that provides the capability to conformance test the Department of Defense suite of upper level protocols including: Internet Protocol (IP) Mil-Std 1777, Transmission Control Protocol (TCP) Mil-Std 1778, File Transfer Protocol (FTP) Mil-Std 1780, Simple Mail Transfer Protocol (SMTP) Mil-Std 1781 and TELNET Protocol Mil-Std 1782.

See p 1

AD-A195 137

17. Document Analysis a. Descriptors

Protocol Test Systems
Conformance Testing
Department of Defense Protocol Suite

b. Identifiers/Open-Ended Terms

Internet Protocol (IP)
Transmission Control Protocol (TCP)
File Transfer Protocol (FTP)
Simple Mail Transfer Protocol (SMTP)
TELNET Protocol

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited Release	UNCLASSIFIED	37
	20. Security Class (This Page) UNCLASSIFIED	22. Price

(See ANSI-Z39.18) See Instructions on Reverse OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce

88 7 06

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

DEFENSE COMMUNICATIONS AGENCY

UPPER LEVEL PROTOCOL TEST SYSTEM

FILE TRANSFER PROTOCOL MIL-STD 1780
REMOTE DRIVER SPECIFICATION

MAY 1988

Disclaimer Concerning Warranty and Liability

This software product and documentation and all future updates to it are provided by the United States Government and the Defense Communications Agency (DCA) for the intended purpose of conducting conformance tests for the DoD suite of higher level protocols. DCA has performed a review and analysis of the product along with tests aimed at insuring the quality of the product, but does not warranty or make any claim as to the quality of this product. The product is provided "as is" without warranty of any kind, either expressed or implied. The user and any potential third parties accept the entire risk for the use, selection, quality, results, and performance of the product and updates. Should the product or updates prove to be defective, inadequate to perform the required tasks, or misrepresented, the resultant damage and any liability or expenses incurred as a result thereof must be borne by the user and/or any third parties involved, but not by the United States Government, including the Department of Commerce and/or The Defense Communications Agency and/or any of their employees or contractors.

Distribution and Copyright

Comments

Comments or questions about this software product and documentation can be addressed in writing to:

DCA Code R640
1860 Wiehle Ave
Reston, VA 22090-5500
ATTN: Protocol Test System Administrator

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF FIGURES

LIST OF TABLES

1. SCOPE AND PURPOSE

This specification describes the Protocol Test System and the program functions of the File Transfer Protocol (FTP) Remote Driver (RD). Section 2, "The Protocol Test System," summarizes the testing procedures used by the Protocol Test System" contains guidelines for developing and FTP RD on a host where an implementation under test ~~(IUT)~~ resides. The role of the FTP RD in protocol testing is also defined.

2. THE PROTOCOL TEST SYSTEM

2.1 TEST TOOLS

The major components of the Protocol Test System are:

- The test Scenario Language - To enable a tester to specify standardized scripts;
- The Scenario Language Compiler and Libraries - To produce code for automated testing;
- The Protocol Reference Implementation -- To define standard functions for the level being tested; and
- The Drivers -- Central, Surrogate, Slave, and Remote -- To execute tests and to provide communication links.

This test system causes an IUT to perform protocol exchanges with a reference implementation. To generate reproducible results, a script controls the driver that controls each IUT through its upper level interface. A complete test executes several scenarios of prescribed actions that exercise one or more protocol features.

2.2 GENERAL REQUIREMENTS OF THE CENTRAL, SURROGATE, SLAVE, AND REMOTE DRIVERS

The Central Test Driver (CTD), which coordinates and monitors protocol testing, combines the Surrogate and Remote Driver results to determine the success or failure of each implementation. The Surrogate Driver provides the transport mechanism between the CTD, the Slave, and the Remote Drivers.

3. REMOTE DRIVER DESIGN AND PROTOCOL TESTING

The following sections describe the Remote Driver design and explain how all drivers interact to communicate protocol commands or responses in Protcol Test System testing.

3.1 THE COMMAND CHANNEL

All drivers except the RD reside at the Protocol Test System site. The Remote Driver operates as a background process using a Transport protocol level connection, which links the RD at a remote site with the laboratory drivers and the reference and IUT protocols.

To set up the command channel, the RD sets up a passive listen on a well-known port and waits for the Surrogate Driver to perform an active open on that port. The command channel is ready when the Transport connection to the Remote and Slave Drivers has been established, and all drivers have initiated communications with their respective protocols. Figure 1 is a diagram of connection establishment, and Table 1 below gives destinations and port numbers.

Table 1. Destinations and Port Numbers

Destination	Port Number
RD Passive Open Port	1204
IUT Server FTP Passive Open Port	21

3.2 FLOW OF COMMANDS

Numbers representing protocol commands travel over the command channel from the Central Driver, through the Surrogate Driver, to the Slave and Remote Drivers (Figure 2, p. 3-3; Table 3, p. 3-9). The Slave and Remote Drivers translate these numbers into the appropriate commands, then issue the commands. Similarly, responses from the reference and IUT protocols are received by the Slave and Remote Drivers and forwarded over the command channel to the Surrogate and Central Drivers. The CTD determines from the combined test responses whether the IUT has functioned properly and reacts accordingly -- by taking the next appropriate action in the testing process -- and the flow of commands is repeated.

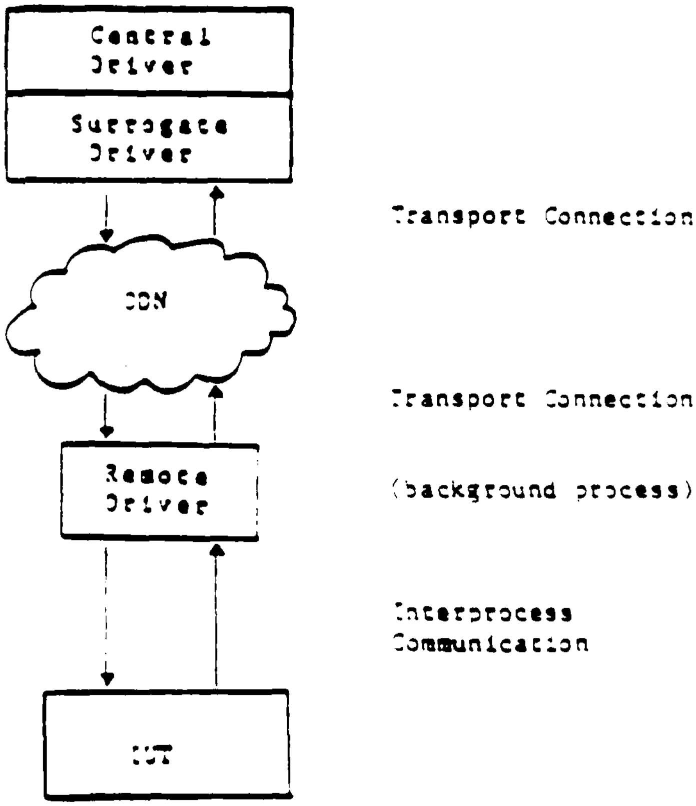


Figure 1. Connection Establishment

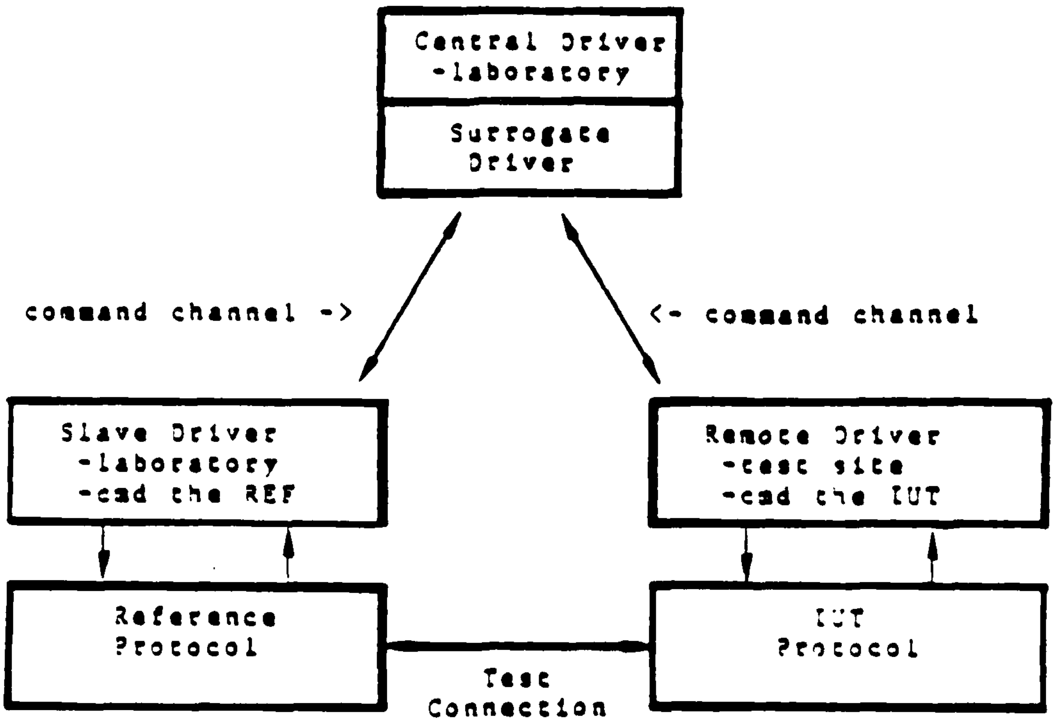


Figure 2. Flow of Commands Between the Four Types of Drivers

3.3 INPUTS AND OUTPUTS

Data is transmitted between the Protocol Test System Central Driver and the Remote Driver in blocks called packets. The laboratory implementation of the Remote Driver uses a PAD (Packet Assembler/Disassembler) function to control the input and output of such packets.

The Remote Driver must be able to send and receive one of three packets: an ACK (Acknowledgment), a NAK (Negative Acknowledgment), or a data packet containing either protocol responses or driver command results. Figure 3 (p. 3-5) shows the generic format of the data packet; Figure C.2 in Appendix C defines a typical data packet in C syntax.

3.3.1 The Control Flag Field

A single octet contains the control flag, which indicates by bit position how the Remote Driver should interpret the rest of the data packet. The bit positions are in ascending order from right to left, as in a DEC (Digital Equipment Corporation) VAX. Figure 4 (p. 3-6) diagrams the bit ordering scheme.

Remote-to-Laboratory Packet and Laboratory-to-Remote Packet

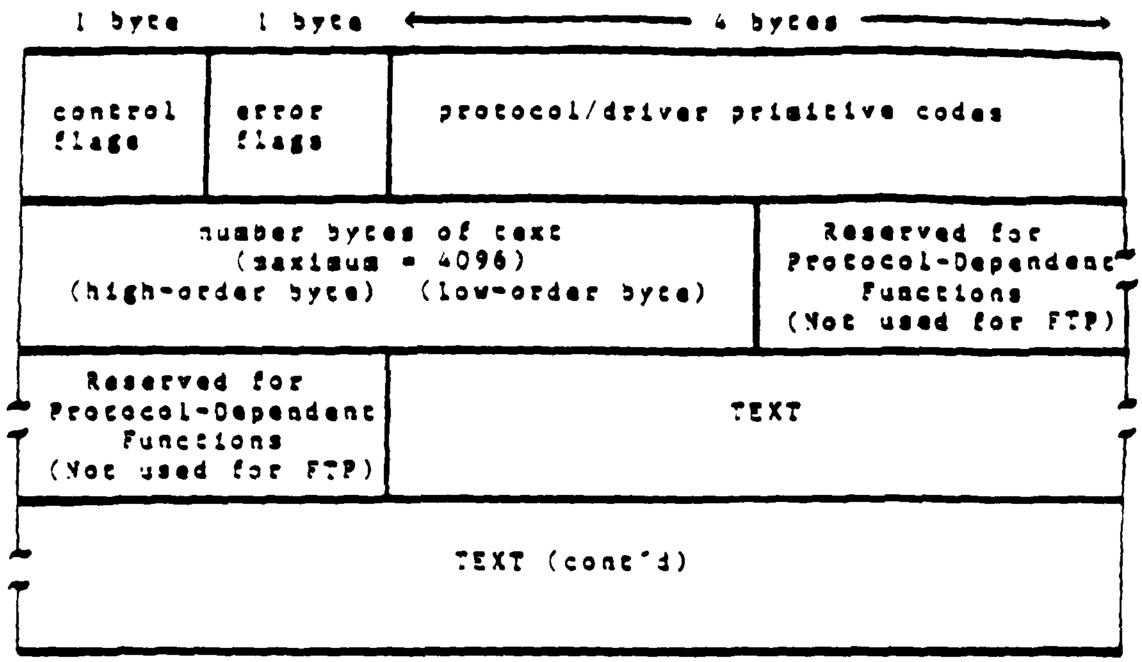
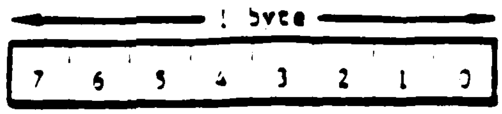


Figure 3.4-1. Generic Format of the Data Packet.

Figure 3. Generic Format of the Data Packet

Control Flag.



bit positions

Figure 4. The Bit Order of a Byte

Table 2 summarizes the configuration of bit positions in the control flag. Only bit positions 0, 1, and 2 are used. The RD sets the bit in position 0 to one (1) to indicate that an ACK packet is being sent. A zero (0) in this position indicates a NAK. If the RD sets the bit in position 1, then a data packet (as opposed to an ACK/NAK packet) is being sent. Setting the DATA bit in a packet could signal either that the RD is sending a protocol response, or that it is sending the results of one of its own driver operations, such as comparing two files. In a file comparison, the data is a character string containing the words "success" or "failure." The bit in position 2 of the first octet of each data packet sent by the CTD determines whether the packet contains a protocol or a driver command.

Table 2. Bit Positions in the Control Flag

Bit Position	Remote⟶Laboratory	Laboratory⟶Remote
0	ACK = 1 NAK = 0	unused
1	data = 1	unused
2	unused	protocol command = 1 driver command = 0
3-7	unused	unused

3.3.2 The Error Flag Field

Error flags will explain the nature of an error occurring at the Remote Driver. Because error codes are not yet implemented, the Protocol Test System makes no attempt to interpret this field.

3.3.3 The Primitive Code Field

The primitive code field can be interpreted in one of two ways -- either as a protocol primitive code or as a driver primitive code. A primitive in this sense means a command that describes some action within the protocol or the driver.

3.3.3.1 The Protocol Primitive Codes

If the control flag indicates a protocol command (i.e., the bit in position 2 is set to 1), then the Remote Driver must translate the integer in the code field to its corresponding protocol primitive according to the descriptions and the code numbers in Table 3 (p. 3-9). What form of data the RD sends to the protocol IUT depends on the IUT's User Interface, but the RD must include this translation capability in its function. To pass qualification testing, an FTP IUT must be able to perform all the primitive actions described in Table 3.

The Remote Driver determines whether arguments are associated with a primitive according to its protocol IUT. If arguments are required for a given primitive, they must be supplied in the "text" field of the data packet. If such arguments are not supplied, then an error condition occurs and the RD must NAK that packet.

The RD can determine whether the required arguments are present by reading the "num_bytes" field, which tells how many bytes of character ASCII

data are in the "text" field. If this field contains a zero, then the RD may assume no arguments have been supplied. However, if the "num_bytes" field contains any positive integer from 1 to 4096, the RD must read the contents of the "text" field and interpret these contents as the primitive's arguments. An integer outside the range of 1 to 4096 means the packet is in error, and that a NAK packet should be returned.

In Table 3 the primitive commands and number codes are followed by one or more argument fields. Primitive names are five or fewer alphabetic characters, and upper- and lowercase characters for both primitives and arguments are treated identically. One or more ASCII spaces separate the primitive code and the argument field, which consists of a character string whose variable length is specified in the num_bytes field. (Section 5.7.1 of MIL-STD-1780 describes FTP commands further.) Three hyphens (---) in the table's "arguments" column indicate that none exist.

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 1 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
user (1)	username	- Causes the User FTP to send the USER command over the command connection. The argument field is the user identification required by the server for access to its file system.
pass (2)	password	- Causes the User FTP to send the PASS command over the command connection. The argument field identifies the user's password.
acct (3)	account_info	- Causes the User FTP to send the ACCT command over the command connection. The argument field identifies the user's account. In most cases this information is optional and host specified.
rein (4)	—	- Causes the User FTP to send the REIN command over the command connection: Reinitializes the FTP session; finishes all pending transactions; then returns the FTP to its default state.
quit (5)	—	- Terminates a USER and, if file transfer is not in progress, the Server closes the command connection and exits. (This specification differs from MIL-STD-1780, which does not require the Server to exit. See the following description of the clos primitive.) If file transfer is in progress, the Server will close the connection after result response. If the User Data Transfer Process (DTP) is transferring files for several USERs but prefers not to close and then reopen connections for each, then the REIN command should be used instead of QUIT.

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 2 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
quit (5) (continued)	---	An unexpected close on a command connection will cause the Server to take the effective action of an abort (ABOR) and a logout (QUIT).
clos (6)	host [port]	- Terminates the User FTP session with the specified host FTP Server, but does not exit. (The clos primitive is the same as the quit primitive, but without the exit.) An optional port number may be supplied; if so, the User FTP will try to terminate the session with the FTP Server at that port.
xport (7)	---	- Causes the User FTP to send an FTP PORT command on the current command connection. The host and port argument for the FTP PORT command is obtained by first issuing a PASV command. (See Section 5.7.2 of MIL-STD-1780.)
pasv (8)	---	- Causes the User FTP to send the PASV command over the command connection. This command requests the Server to "listen" on a data port (not its default data port) and to wait for a connection rather than to initiate one upon receipt of a transfer command. The response to the PASV command includes the host and port address on which the Server is listening.
type (9)	type_code	- Causes the User FTP to send the TYPE command over the command connection. The argument indicates the data representation type as specified in the Section on Data Representation and Storage in MIL-STD-1780.

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 3 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
stru (10)	struct_code	- Causes the User FTP to send the STRU command over the command connection. Indicates the structure of the file being transferred. The argument specifies a structure of file, page, or record. (See Section 5.7.2 of MIL-STD-1780.)
mode (11)	mode_code	- Causes the User FTP to send the MODE command over the command connection. Indicates the transmission mode of the file being transferred. The argument specifies the data transfer mode of stream, block, or compressed. (See Section 5.7.2 of MIL-STD-1780.)
retr (12)	pathname	- Causes the User FTP to send the RETR command over the command connection. This command causes the Server to initiate the transfer of a copy of the file, specified in the pathname argument, to the Server or the User over the data connection.
stor (13)	pathname	- Causes the User FTP to send the STOR command over the command connection. This command causes the Server to accept the data transferred over the data connection and to store the data as a file, specified in the pathname argument, at the Server's site.
appe (14)	pathname	- Causes the User FTP to send the APPE command over the command connection. This command causes the Server to accept the data transferred over the data connection and to store the data in a file, specified by the pathname argument, at the Server site. If no such file exists,

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 4 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
appe (14) (continued)	pathname	then a new file is created at the Server site.
allo (15)	decimal_int [R decimal_int]	- Causes the User FTP to send the ALLO command over the command connection. Causes the Server to reserve sufficient storage to accommodate the new file to be transferred. The argument indicates the number of bytes to be reserved. This command may not be required by those Servers that require no prior declaration of maximum file size, and should be treated as a NOOP in such cases. (See Section 5.7.3 in MIL-STD-1780 for complete requirements.)
rnfr (16)	pathname	- Causes the User FTP to send the RNFR command over the command connection. This command, which identifies the file to be renamed using the pathname argument, must be followed immediately by a "rename to" command (RNTO) specifying the new file pathname.
rnto (17)	pathname	- Causes the User FTP to send the RNTO command over the command connection. This command specifies the new pathname of the file identified in the immediately preceding "rename from" command. Together, the two commands cause a file to be renamed.
abor (18)	---	- Causes the User FTP to send the ABOR command over the command connection. This command tells the Server to abort the previous FTP service command and any

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 5 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
abor (18) (continued)	---	associated transfer of data. The abort command requires special action to force recognition by the Server, as discussed in the MIL-STD-1780 section on FTP commands. No action is to be taken if the previous command has been completed (including data transfer). The TELNET connection is not to be closed by the Server, but the data connection must be closed. When this command is received, there are two cases for the Server: the FTP service command was already completed, or the FTP service command is still in progress. In the first case, the Server closes the data connection (if it is open), and responds with a 226 reply indicating the abort command was successfully processed. In the second case, the Server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate abnormal service request termination. The Server then sends a 226 reply to indicate the abort command was successfully processed.
dele (19)	pathname	- Causes the User FTP to send the DELE command over the command connection. This command causes the file specified in the pathname argument to be deleted at the Server site. If an extra level of protection is desired, it should be provided by the User FTP process.
cwd (20)	pathname	- Causes the User FTP to send the CWD command over the command connection. This command allows the user to work with a different directory or dataset for file

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 6 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
cwd (20) (continued)	pathname	storage or retrieval without altering his login or accounting information. The argument is a pathname specifying a directory or other system-dependent file group designator.
list (21)	pathname	- Causes the User FTP to send the LIST command over the command connection. This command causes a list to be sent from the Server to the Passive Data Transfer Process (DTP). If the pathname specifies a directory, the Server should transfer a list of files in the specified directory. If the pathname specifies a file, then the Server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or EBCDIC.
nlst (22)	pathname	- Causes the User FTP to send the NLST command over the command connection. This command causes a directory listing to be sent from the Server to the User site. The pathname should specify a directory or other system-dependent file group descriptor. A null argument implies the current directory. The Server returns a stream of filenames and no other information. The data is transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by Carriage Return-Linefeed (<CRLF>) or Newline (<NL>). The user must ensure the TYPE is correct.

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 7 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
site (23)	string	- Causes the User FTP to send the SITE command over the command connection. The Server uses this command to provide services specific to its system essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the REMO (remote help) SITE command.
stat (24)	pathname	- Causes the User FTP to send the STAT command over the command connection. This command causes a status response to be sent over the command connection in the form of a reply.
remo (25)	string	- Causes the User FTP to send the HELP command over the command connection. This command causes the Server to send helpful information regarding its implementation status over the command connection to the user. The command may take an argument (e.g., any command name and return more specific information as a response. The reply is type 211 or 214. (See Section 5.7.3 of MIL-STD-1780.)
noop (26)	—	- Causes the User FTP to send the NOOP command over the command connection. This command does not affect any parameters or previously entered commands. It specifies no action except that the Server must send an OK reply.
get (27)	remote-file [local-file]	- Retrieves the remote-file and stores it on the local machine. If the local-file

NOTE: Commands 27 through 32 have no equivalent in MIL-STD-1780. They were created as a superset of standard FTP commands to support driver and reference functions in the Protocol Test System.

Table 3. Protocol Primitive Commands, Number Codes, and Arguments, with Consequent FTP IUT Action (Page 8 of 8)

Primitive (No. Code)	Argument	FTP IUT Action
get (27) (continued)	remote-file [local-file]	name is not specified, the file is given the same name it has on the remote machine. The get primitive translates into the PORT and RETR FTP commands as specified in MIL-STD-1780.
put (28)	local-file [remote-file]	- Stores the local-file on the remote machine. If remote-file name is left unspecified, the local-file name is used. The put primitive translates into the PORT and STOR FTP commands as specified in MIL-STD-1780.
open (29)	host [port]	- Establishes a connection to the specified host FTP Server. An optional port number may be supplied, in which case the User FTP will attempt to contact an FTP Server at that port.
swit (30)	host [port]	- Switches to any specified host that has already been connected. A port number may be supplied for any connection already opened using the open command with port argument. This command does not send any data across the command connection.
die (31)	—	- Terminates the FTP session with the remote Server (an ungraceful close) and exits User FTP. This command does not send any data across the command connection.
quot (32)	arg1 arg2	- Sends the arguments arg1 and arg2 to the remote FTP Server over the command connection. One or more FTP reply codes are expected in return.

3.3.3.2 The Driver Primitive Codes

If the control flag indicates a driver command (the bit in position 2 is set to zero), the Remote Driver must translate the integer contained in the code field to its corresponding driver primitive (Table 4). The RD then performs the appropriate action, as specified in the following sections.

Table 4. The Driver Primitives

Code	Action
0	- Kill the Remote Driver process.
1	- Send associated data to the IUT.
2	- Compare two files in local file system and report if different.

3.3.3.2.1 The KILL Driver Primitive. If the Remote Driver interprets a driver primitive code of 0, then after ACKing the driver primitive packet, the process must find some way to terminate itself. No other action is necessary. The Remote Driver is not responsible for conducting a graceful shutdown of the protocol; it is the responsibility of the Central Driver to cause FTP to quit. The RD also is not required to shut itself down gracefully, although this action is encouraged.

3.3.3.2.2 The DATA Driver Primitive. This command is used when testing requires large amounts of text to be passed. If the Remote Driver receives a driver primitive code of 1, indicating the DATA command, then after ACKing the driver primitive packet, the RD must send the text portion of the data packet to the IUT as data. The other fields can be ignored.

3.3.3.2.3 <u>The COMPARE Driver Primitive</u>. The COMPARE driver primitive is used to test whether the FTP IUT has correctly implemented the different modes of file transfer. When the Remote Driver receives a driver primitive code of 2, indicating the COMPARE driver command, then after ACKing the driver primitive packet, the Remote Driver must perform a compare operation on the two files named in the text field as arguments of this command. If the files are found and successfully opened, then they must be read and compared byte-by-byte. If the two files are exactly alike, then the Remote Driver must return a data packet containing the ASCII equivalent of the character string "success" in the text field. If the two files differ by even a bit, however, the ASCII equivalent of the character string "failure" is returned to the Central Driver in a data packet. If an error condition occurs (e.g., the files are nonexistent or cannot be opened), then the Remote Driver must also send back the "failure" string. When a character string is sent back in a data packet, the fields should contain the following values:

Code:	0	(indicating a driver command);
cntl_flag:	1	(indicating DATA);
num_bytes:	7	(both strings have 7 bytes, and here the null-byte string terminator is not necessary);

text: "success"
or
"failure" (a 7-byte ASCII character string).

3.4 ACK/NAK PACKETS

The Remote Driver is required to acknowledge the receipt of every driver or protocol command (ACK = positive acknowledgment; NAK = negative acknowledgment). When it is able to read and interpret a packet from the TCP

connection, the RD sends an ACK packet. If it detects an error, however, the Remote Driver responds with a NAK packet. The RD also sends a NAK packet if a read is successful but a code is unknown, or some other field is in error.

The following values indicate ACK or NAK packets:

ACK - code: 0

cntl_flag: bit position 0 set to one (1)
bit position 2 set to one (1) if protocol command,
to zero (0) if driver command

num_bytes: 0

text: 0

NAK - code: 0

cntl_flag: bit position 0 set to zero (0)
bit position 2 set to one (1) if protocol command,
to zero (0) if driver command

num_bytes: 0

text: 0

3.5 TIMING

A Remote Driver has no timing constraints per se, but it should not add considerably to a protocol IUT's response time. For example, if the CTD does not receive a data packet within the time specified in a script, then a timeout condition will occur. Such a condition could cause an IUT to fail the test.

3.6 FLEXIBILITY

Although drivers have no special flexibility requirements, adaptable hardware and software enhances their operation and expandability. In the next phase of protocol testing, they may need to command passive recorders or other devices. Drivers thus should be able to add new commands or to interpret unused header bytes.

APPENDIX A - References

"Military Standard File Transfer Protocol" (MIL-STD-1780); May 1984; Department of Defense.

"Military Standard TELNET Protocol" (MIL-STD-1782); August 1983; Department of Defense.

System Development Corporation, "Laboratory Implementation Plan," TM-WD-8574/000/02, January 1985.

System Development Corporation, "Laboratory Specification," TM-WD-7172/520/00, August 1984.

System Development Corporation, "Higher Level Capability Plan," TM-WD-8573/000/00, April 1984.

System Development Corporation, "FTP Test Tools User's Manual," TM-WD-8601/003/02, November 1985.

System Development Corporation, "FTP Test Tools Programmer's Manual," TM-WD-8601/004/02, November 1985.

Kernighan, B. W., and Ritchie, D. M.; The C Programming Language; Prentice-Hall, Inc.; Englewood Cliffs, NJ; 1978.

Kernighan, B. W., and Pike, R.; The UNIX Programming Environment; Prentice-Hall, Inc.; Englewood Cliffs, NJ; 1984.

APPENDIX B - GLOSSARY

ACK (Acknowledgment)

In data transfer between devices, data is blocked into units of a size given in each block's header. If the received data is found to be without errors, then the receiving device sends an ACK block back to the transmitting unit to acknowledge receipt. The transmitting device then sends the next block. If the receiving unit detects errors, however, it sends a NAK block to indicate the received data contained errors.

ASCII (American Standard Code for Information Interchange)

A standard code for the representation of alphanumeric information. ASCII is an 8-bit code in which 7 bits indicate the character represented and the 8th, high-order bit is used for parity.

CTD (Central Test Driver)

DDN (Defense Data Network)

A U.S. military packet-switched network.

DEC (Digital Equipment Corporation)

DoD (Department of Defense)

DTP (Data Transfer Process)

The File Transfer Protocol (FTP) has two different process states for data transfer. The Server-DTP (the active state) establishes the data connection with the listening data port, sets up parameters for transfer and storage, and transfers data on command from its protocol interpreter. The User-DTP (the passive state) listens for a connection on the data port from a Server-FTP process.

EBCDIC (Extended Binary-Coded Decimal Interchange Code)

An 8-bit code for representing alphanumeric information within a computer. EBCDIC is most widely used in large computer systems. See also <u>ASCII</u>.

EDN (Exploratory Data Network)

FTP (File Transfer Protocol)

The DoD interim standard protocol used to transfer data reliably between hosts on DoD packet-switched networks.

IUT (Implementation Under Test)

A given vendor's protocol implementation and the subject of the immediate test.

MIL-STD (Military Standard)

Specification published by the DoD.

NAK (Negative Acknowledgment)

In data transfer between devices, a NAK block is returned by the receiving device to the sending device to indicate the preceding data block contained errors. See also ACK.

packet switching

A method of transmitting messages through a network in which long messages are subdivided into short packets. The packets are then transmitted as in message switching.

PAD (Packet Assembler/Disassembler)

In this document, PAD refers to a module of a structured program responsible for reading and writing data packets. Not to be confused with the other known usage, which describes a device to provide service to asynchronous terminals within an X.25 network.

PAR (Positive Acknowledgment and Response)

A simple communication protocol stating that every packet received must be either ACKed or NAKed.

protocol

A set of rules governing the operation of functional units to achieve communication.

TCP (Transmission Control Protocol)

The DoD standard connection-oriented transport protocol used to provide reliable, sequenced, end-to-end service.

Telnet

The DoD interim standard Virtual Terminal Protocol for the flexible attachment of remote terminals to host computer systems over the network.

APPENDIX C - Examples of Remote Driver Implementation in UNIX/C

C-1 CONNECTION ESTABLISHMENT

The Remote Driver must be able to establish interprocess communication; i.e., to read and write data stream from a Transport (TCP) socket or from the FTP IUT. Although the method of interprocess communication is not specified for the Remote Driver, a description of how a Remote Driver is implemented may be helpful. (See Figure C.1).

The Protocol Test System runs in a UNIX 4.2 BSD environment. The lab's Remote Driver is implemented in C language, which provides access to several interprocess communication system calls (e.g., fork, socket, bind, pipe, listen, and accept system calls).

Specifically, the IUT process is activated by the Remote Driver process using the fork system call. This forking causes a parent-child relathionship to be formed between the two processes. Each process then determines whether it is the child or the parent. If the process is the parent, then it exits.

In effect, this action leaves the child process running as a background process. This background process then sets up the TCP socket connection by listening passively on the TCP port specified in Table 1.

After the TCP connection is established, the Remote Driver sets up communication with the IUT by forking another process and setting up interprocess communication through the use of the pipe system call. Once the pipes are established, the input/output of the two processes can be manipulated so that the two processes end up communicating with each other. Since the parent and child processes are exactly the same (except for process identification numbers) and would be useless talking to themselves, the protocol IUT is overlayed onto the child process using the exec system call.

When this procedure is successfully completed, the Remote Driver -- running as a background process -- is receiving and sending data over the TCP connection on one side, and sending/receiving data to/from the protocol IUT on the other side. (See Figure 1, p. 3-2.)

C.2 A PAD FUNCTION

A Packet Assembler/Disassembler (PAD) function is useful for implementing a Remote Driver. Because the two basic functions of receiving and transmitting packets are performed repeatedly, it may be wise to modularize them. When the Remote Driver reads data from the command channel, it must be able to interpret the bytes correctly. In a UNIX/C implementation, the method used to achieve this result is to load the data into a data structure where distinct fields can be declared. In C, this is the "struct" declaration. Each collection of fields can then be referenced as a whole. The term "packet" has been used in this document as the name of this data structure reference. The "struct" declaration is shown in Figure C.2 (p. C-4).

```
*                 CONNECTION ESTABLISHMENT 4.2BSD UNIX/C                  */

                                /* spawn IUT process, and                 */
                                /* put this process into background...    */
                                /* parent returns pid; child returns 0    */
if ( (fork_stat = fork()) > 0 )                                 /* pid > 0*/
   exit( );            /*  so, if parent, then exit.                      */
                       /* else, this is child, so continue!               */
pp = getprotobyname("tcp");   /* 4.2 BSD system call                     */

while ((s = socket (AF_INET, SOCK_STREAM, pp->p_proto)) < 0)
{  if  (errno != 0)
   {   perror ("FTP_SD:  socket");
       sleep (1);
   }
{
sin.sin_port = sp->s_port;          /* set port #                         */
                                    /* bind name to socket                */
while (bind (s, (caddr_t)&sin, sizeof (sin), 0) < 0)
{  perror ("FTP_SD:  bind");
   sleep(10);
}
listen (s, 10);             /* passively listen on socket                 */
                                          /* accept connection           */
s2 = accept (s, (caddr_t)&from, &fromlen);
                                          /* set up pipes                 */
if (pipe(fd1) < 0)
   return (-1);
if (pipe(fd2) < 0)
   return (-1);
in_pipe = fd1[0];
out_pipe = fd2[1];
if (fork( ) == 0)           /* if child then execute the following       */
{  close (0);                         /* close std input                  */
   if (dup2 (fd2[0], 0) < 0 )         /* open READ side of pipe           */
      perror ("dup2");
close (1);                            /* close std output                 */
if (dup2 (fd1[1], 1) < 0)             /* open WRITE side                  */
execl ("/usr/iut/iut_ftp",  "iut_ftp", "-v", "-n", NULL);
}
```

Figure C.1. Outline of Connection Establishment in 4.2 BSD UNIX/C

```
#define  MAX_TEXT_LEN        4096
struct remote_pack  {
                char cntl_flag;
                char err_flag;
                int  code;
                int  num_bytes;
                int  reserved;
                char text[MAX_TEXT_LEN];
              };
```

Figure C.2. The C Syntax Format of the Data Packet

The reception mode of the PAD function reads data and "packets" the data. The PAD accomplishes this task by reading the first 14 bytes of data from the input stream. This first 14 bytes is effectively the header of the data packet. It contains all the information needed to process the packet, including the integer in the "num_bytes" field that indicates the number of bytes of character text to follow -- if any.

The transmission mode of the PAD function unpackets the data and sends data over the communication channel. The PAD accomplishes this task by reading the header of the packet and writing the information into a memory space allocated for a large character string. The size is equal to 4096 bytes -- the maximum text size -- plus 14 bytes for the header information. If the text size is less than the maximum, then only that much need be written. The Remote Driver must transmit the data as a normal byte stream. If the input data stream is correctly packeted into a data structure, then interpretation of the fields in the packet should become clearer and less susceptible to error. An example of a PAD implemented in C is given in Figure C.3 (p. C-5).

```
#define HEADER_SIZE    14
#define MAX_TEXT_LEN  4096
#define MAX_PACK_LEN HEADER_SIZE+MAX_TEXT_LEN

send_packet(packet,sock)       /* packet disassembler      */
struct remote_pack *packet;
int sock;
{    register int i;
     char send_buffer[MAX_PACK_LEN];

     send_buffer[0] = packet->cntl_flag;
     send_buffer[1] = packet->err_flag;
     *((int *)(send_buffer+2))     = packet->code;
     *((int *)(send_buffer+6))     = packet->num_bytes;
     *((int *)(send_buffer+10))    = packet->reserved;
     for(i=0;i<packet->num_bytes;i++)
          send_buffer[i+14] = packet->text[i];
                              /* send the packet           */
     return(write(sock, send_buffer,
                 packet->num_bytes+HEADER_SIZE));
}

recv_packet(packet,sock)        /* packet assembler        */
struct remote_pack *packet;
int sock;
{
     char recv_buffer[HEADER_SIZE];
     int result;
                              /* read the header          */
     if   ( (result = read(sock, recv_buffer, HEADER_SIZE))  !=
                                                    HEADER_SIZE)
     {
            return (result);
     }
     packet->cntl_flag = recv_buffer[0];
     packet->err_flag  = recv_buffer[1];
     packet->code    = *((int *) (recv_buffer+2));
     packet->num_bytes = *((int *) (recv_buffer+6));
     packet->reserved  = *((int *) (recv_buffer+10));
                              /* read the text            */
     if (packet->num_bytes)
     {
       if(read(sock, packet->text, packet->num_bytes)  !=
                                           packet->num_bytes)
         return(-1);
     }
     return(0);
}
```

Figure C.3. A Packet Assembler/Disassembler in C